GOVERNMENT COLLEGE OF ENGINEERING, KALAHANDI



DEPARTMENT OF ELECTRICAL ENGINEERING

Lecture notes on Applied Deep Learning

Prepared by Dr. Soudamini Behera, Associate Professor,Dept. of Electrical Engineering, GCE, Kalahandi

ACC: ACC 3

ACC Bucket : Python & Machine Learning - 3

Course: Applied Deep Learning

OVERALL COURSE OBJECTIVES: The objective of this course is to synthesize comprehensive deep learning knowledge, cultivate capabilities to configure and optimize neural networks, refine machine learning troubleshooting strategies, and apply advanced concepts, all while actively contributing to the development of leading-edge AI technology.

LEARNING OUTCOMES: On successful completion of the course the students shall be able to:

1. Construct, train, and implement fully connected deep neural networks and apply them to various applications.

2. Evaluate and employ standard neural network techniques such as initialization, L2 and dropout regularization, hyperparameter tuning, batch normalization, and gradient checking.

3. Implement a variety of optimization algorithms, including mini-batch gradient descent, Momentum, RMSprop and Adam, and check for their convergence.

4. Implement a neural network in TensorFlow.

5. Diagnose errors in a machine learning system and strategize solutions for reducing these errors.

6. Apply advanced machine learning concepts such as end-to-end learning, transfer learning,and multi-task learning.

COURSE CONTENT:

Module 1: Neural Networks and Deep Learning [25 Hours]

Sub-Topic

Supervised Learning with Neural Networks

Binary Classification

Derivatives with a Computation Graph

Vectorizing Logistic Regression

Shallow Neural Networks

Building Blocks of Deep Neural Networks

Module 2: Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization [24 Hours]

Sub-Topic

Practical Aspects of Deep Learning

Understanding Exponentially Weighted Averages

Weight Initialization for Deep Networks

The Problem of Local Optima

Optimization Algorithms

Hyperparameter Tuning, Batch Normalization and Programming Frameworks

Module 3: Structuring Machine Learning Projects [7 Hours]

Sub-Topic

Why ML Strategy

Understanding Human-level Performance

Surpassing Human-level Performance

Transfer Learning

Multi-task Learning

What is End-to-end Deep Learning?

Disclaimer

This document does not claim any originality and cannot be used as a substitute for prescribed textbooks. The information presented here is merely a collection by the committee members for their respective teaching assignments. Various sources as mentioned at the end of the document as well as freely available material from internet were consulted for preparing this document.

The ownership of the information lies with the respective authors or institutions. Further, this document is not intended to be used for commercial purpose and the committee members are not accountable for any issues, legal or otherwise, arising out of use of this document. The committee members make no representations or warranties with respect to the accuracy or completeness of the contents of this document and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. The committee members shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

Module 1: Neural Networks and Deep Learning

1. Introduction to Neural Networks

A **neural network** is a machine-learning model inspired by the way biological neurons process information. It consists of interconnected computational units called **neurons**, arranged in layers.

A typical neural network contains:

- **Input layer** – receives input features.
- **Hidden layer(s)** – performs intermediate computations.
- **Output layer** – produces the final prediction.

Basic Structure

Input Layer→ Hidden Layer(s)→Output Layer

For an input vector

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

a neuron calculates a weighted sum:

$$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

or

$$z = W^T X + b$$

where:

- X = input vector
- W = weight vector
- b = bias
- z = weighted input

An activation function is then applied:

$$a = g(z)$$

where g is the activation function.

2. Supervised Learning with Neural Networks

Supervised learning is a machine-learning approach in which the model learns from labelled training data.

The training dataset consists of:

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

where:

- $x^{(i)}$ = input of the i^{th} training example
- $y^{(i)}$ = corresponding target/output
- m = number of training examples

The objective is to learn a function:

$$\hat{y} = f(x)$$

such that the predicted output $\hat{y}$ is as close as possible to the actual output y.

Applications

Neural networks are widely used for:

- Image classification
- Speech recognition
- Face recognition
- Medical diagnosis
- Fraud detection
- Natural language processing
- Predictive maintenance
- Autonomous vehicles

3. Binary Classification

Binary classification is a classification problem where the output has only two possible classes.

Examples:

- Disease / No disease
- Spam / Not spam

- Pass / Fail
- Yes / No
- Defective / Non-defective

Usually the output is represented as:

$$y \in \{0, 1\}$$

Logistic Regression Model

For binary classification, the sigmoid function is commonly used:

$$\hat{y} = a = \sigma(z)$$

where

$$z = w^T x + b$$

and

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

The sigmoid function produces an output between 0 and 1:

$$0 < \sigma(z) < 1$$

Therefore, it can be interpreted as a probability.

Decision Rule

For example:

$$\hat{y} = \begin{cases} 1, & \text{if } a \geq 0.5 \\ 0, & \text{if } a < 0.5 \end{cases}$$

4. Logistic Regression Cost Function

For a single training example, the logistic regression loss is:

$$L(\hat{y}, y) = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right]$$

For m training examples, the cost function is:

$$J(w, b) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})\right]$$

The objective of training is to minimize $J(w, b)$.

5. Computation Graph

A **computation graph** represents mathematical operations as a sequence of connected nodes.

Consider:

$$z = w^T x + b$$

followed by

$$a = \sigma(z)$$

and finally the loss:

$$L(a, y)$$

The computational flow is:

$$x, w, b \to z \to a \to L$$

Forward Propagation

The calculations proceed from input toward output:

$$z = w^T x + b$$

$$a = \sigma(z)$$

$$L = -[y \log a + (1 - y) \log(1 - a)]$$

This is called **forward propagation**.

Backward Propagation

The derivatives of the loss are calculated by moving backward through the computation graph.

This is the fundamental principle behind **backpropagation**.

6. Derivatives and Gradient Descent

The derivative measures how much a function changes when its input changes.

For a function:

$$y = f(x)$$

the derivative is:

$$\frac{dy}{dx}$$

In neural networks, derivatives are used to determine how the parameters should be changed to reduce the loss.

Gradient Descent

The basic parameter update is:

$$w := w - \alpha \frac{\partial J}{\partial w}$$

$$b := b - \alpha \frac{\partial J}{\partial b}$$

where:

- α = learning rate
- J = cost function

A suitable learning rate is important.

If α is:

- **Too small** → training becomes very slow.
- **Too large** → training may become unstable or fail to converge.

7. Vectorizing Logistic Regression

Vectorization means performing operations on complete vectors or matrices instead of using explicit loops for every training example.

For a dataset:

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

the predictions can be calculated as:

$$Z = W^T X + b$$

and

$$A = \sigma(Z)$$

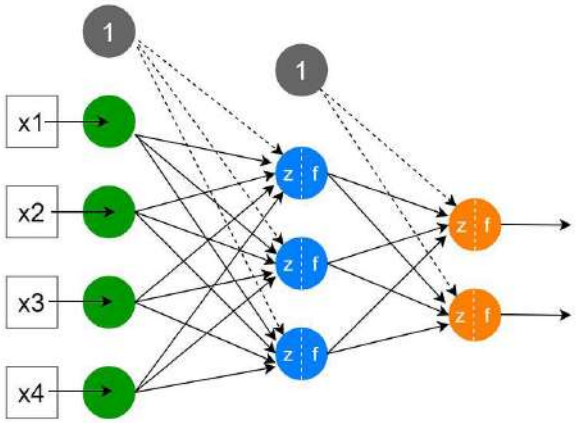
Vectorization significantly improves computational efficiency, particularly when training deep neural networks on large datasets.

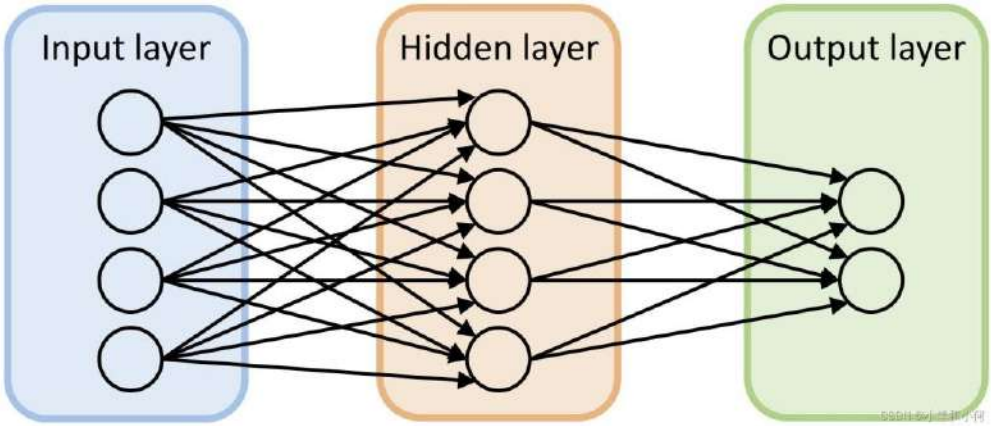
Advantages

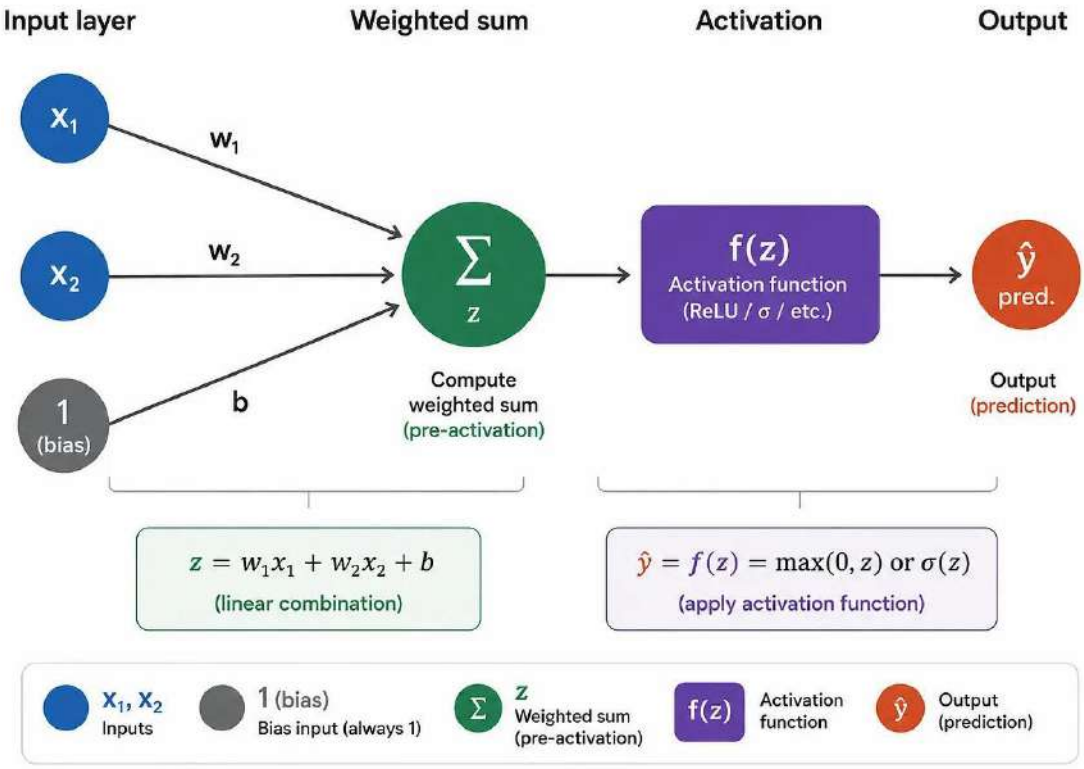
- Faster computation
- Better use of hardware
- Efficient matrix operations
- Reduced programming complexity
- Suitable for GPU-based computation

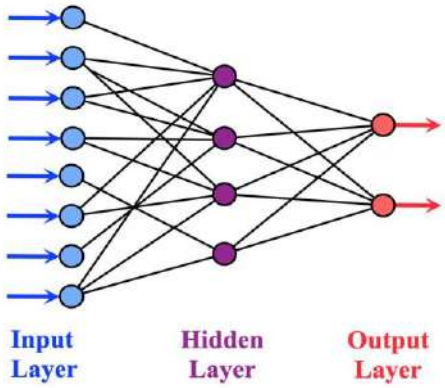
8. Shallow Neural Networks

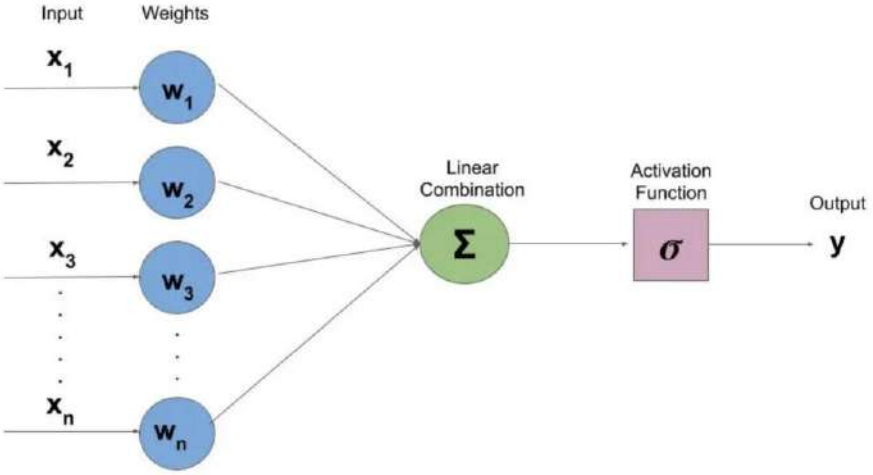
A shallow neural network generally contains one hidden layer between the input and output layers.






Input layer
Weighted sum
Activation
Output
x1
x2
1
(bias)
w1
w2
b
Σ
z
Compute
weighted sum
(pre-activation)
f(z)
Activation function
(ReLU / σ / etc.)
ŷ
pred.
Output
(prediction)
z = w1x1 + w2x2 + b
(linear combination)
ŷ = f(z) = max(0, z) or σ(z)
(apply activation function)
x1, x2
Inputs
1 (bias)
Bias input (always 1)
z
Weighted sum
(pre-activation)
f(z)
Activation
function
ŷ
Output
(prediction)


Input
Layer
Hidden
Layer
Output
Layer


Input
Weights
x1
x2
x3
xn
w1
w2
w3
wn
Linear
Combination
Σ
Activation
Function
σ
Output
y

For layer l:

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$

and

$$A^{[l]} = g^{[l]}(Z^{[l]})$$

where:

- $A^{[l-1]}$ = activation from previous layer
- $W^{[l]}$ = weight matrix
- $b^{[l]}$ = bias vector
- $Z^{[l]}$ = linear transformation
- $A^{[l]}$ = activation of current layer

For the input:

$$A^{[0]} = X$$

The final output is:

$$\hat{Y} = A^{[L]}$$

where L is the number of layers.

11. Forward Propagation in a Deep Neural Network

For a network with L layers:

Layer 1

$$Z^{[1]} = W^{[1]} X + b^{[1]}$$

$$A^{[1]} = g^{[1]}(Z^{[1]})$$

Layer 2

$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$

$$A^{[2]} = g^{[2]}(Z^{[2]})$$

Continue until the output layer:

$$Z^{[L]} = W^{[L]} A^{[L-1]} + b^{[L]}$$

$$A^{[L]} = g^{[L]}(Z^{[L]})$$

Thus:

$$X \to A^{[1]} \to A^{[2]} \to \cdots \to A^{[L]}$$

12. Backpropagation

Backpropagation is an algorithm used to calculate gradients of the loss function with respect to the network parameters.

The process consists of:

1. Forward propagation
2. Calculation of loss
3. Backward propagation
4. Parameter update

Using the chain rule:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$$

The gradients are propagated from the output layer toward the input layer.

The weights are then updated using gradient descent:

$$W^{[l]} = W^{[l]} - \alpha \frac{\partial J}{\partial W^{[l]}}$$

and

$$b^{[l]} = b^{[l]} - \alpha \frac{\partial J}{\partial b^{[l]}}$$

13. Training a Neural Network

The general training process is:

Step 1: Initialize parameters.

Step 2: Perform forward propagation.

Step 3: Calculate the loss.

Step 4: Perform backpropagation.

Step 5: Calculate gradients.

Step 6: Update weights and biases.

Step 7: Repeat for several iterations/epochs.

The objective is to minimize the training loss and obtain good generalization on unseen data.

14. Neural Network Hyperparameters

Important parameters that influence neural-network training include:

- Number of hidden layers
- Number of neurons per layer
- Learning rate
- Number of training epochs
- Activation functions
- Batch size
- Optimization algorithm

These are generally called **hyperparameters** because they are selected before or during the training process rather than learned directly as ordinary network weights.

15. Practical Example: Binary Classification

Suppose a neural network predicts whether a student will pass an examination based on:

- x_1 = study hours
- x_2 = attendance percentage

The input vector is:

$$X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

The network produces:

$$z = w_1 x_1 + w_2 x_2 + b$$

Then:

$$\hat{y} = \frac{1}{1 + e^{-z}}$$

If:

$$\hat{y} \geq 0.5$$

the student may be classified as **Fail**.

16. Important Numerical Example

Consider:

$$w = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad b = -1$$

Calculate z:

$$z = w^T x + b$$

$$z = (2)(1) + (1)(2) - 1$$

$$z = 3$$

Now calculate the sigmoid output:

$$\hat{y} = \frac{1}{1 + e^{-3}}$$

$$\boxed{\hat{y} \approx 0.9526}$$

Therefore, using a threshold of 0.5:

$$\boxed{\text{Predicted class} = 1}$$

17. Important Terms

Term	Meaning
Neuron	**Basic computational unit of a neural network**
Weight	**Parameter representing the importance of an input**
Bias	**Adjustable parameter added to the weighted sum**
Activation	**Output of a neuron after applying an activation function**
Epoch	**One complete pass through the training dataset**
Learning Rate	**Step size used during parameter updates**
Forward Propagation	**Computing output from input to output**
Backpropagation	**Computing gradients from output toward input**
Loss Function	**Measures prediction error**
Gradient	**Direction/rate of change of the loss**

Deep Learning	**Learning using neural networks with multiple layers**
Vectorization	**Efficient computation using vectors/matrices**

18. Key Formulas for Module 1

Linear Transformation

$$Z = WX + b$$

Sigmoid

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

ReLU

$$ReLU(z) = \max(0, z)$$

Binary Cross-Entropy Loss

$$L = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

Cost Function

$$J = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

Gradient Descent

$$W := W - \alpha \frac{\partial J}{\partial W}$$

$$b := b - \alpha \frac{\partial J}{\partial b}$$

Deep Network Layer

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$

$$A^{[l]} = g^{[l]}(Z^{[l]})$$

Module 1 – Short-Answer Questions with Answers

Applied Deep Learning

1. What is supervised learning?

Answer:
Supervised learning is a machine-learning technique in which a model learns from **labelled training data**. Each input is associated with a known output or target. The model learns a mapping between input and output and uses it to predict outputs for new data.

Example: Predicting whether an email is **spam or not spam**.

2. What is a neural network?

Answer:
A neural network is a machine-learning model composed of interconnected computational units called **neurons**. These neurons are organized into an input layer, one or more hidden layers, and an output layer. Neural networks learn patterns by adjusting their weights and biases during training.

3. Define a neuron, weight, and bias.

Answer:

- **Neuron:** A computational unit that receives inputs, calculates a weighted sum, and applies an activation function.
- **Weight:** A parameter that determines the importance of an input.
- **Bias:** An adjustable parameter added to the weighted sum to shift the activation.

The basic equation is:

$$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

4. What is binary classification?

Answer:
Binary classification is a supervised-learning problem in which the output belongs to one of **two possible classes**.

Examples:

- Pass / Fail
- Spam / Not Spam
- Disease / No Disease
- Yes / No

The output is generally represented as:

$$y \in \{0, 1\}$$

5. Define the sigmoid function.

Answer:
The sigmoid function is an activation function that converts any real-valued input into a value between **0 and 1**.

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

It is commonly used in the output layer of a neural network for **binary classification**.

Its derivative is:

$$\sigma'(z) = \sigma(z)[1 - \sigma(z)]$$

6. What is a computation graph?

Answer:
A computation graph is a graphical representation of a mathematical computation in which variables and operations are represented as connected nodes.

For example:

$$x, w, b \to z \to a \to L$$

where:

$$z = w^T x + b$$

$$a = \sigma(z)$$

and L is the loss.

Computation graphs are useful for performing **forward propagation and backpropagation**.

7. What is forward propagation?

Answer:
Forward propagation is the process of passing input data through the neural network from the **input layer to the output layer** to obtain a prediction.

For a layer ll:

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$

$$A^{[l]} = g^{[l]}(Z^{[l]})$$

The final activation gives the predicted output.

8. What is backpropagation?

Answer:
Backpropagation is an algorithm used to calculate the **gradients of the loss function with respect to the weights and biases** of a neural network.

It works backward from the output layer toward the input layer using the **chain rule of differentiation**.

The calculated gradients are then used by an optimization algorithm such as gradient descent to update the network parameters.

9. What is vectorization?

Answer:
Vectorization is the process of performing mathematical operations on **vectors or matrices instead of using individual loops** for each calculation.

For example:

$$Z = W^T X + b$$

Vectorization makes neural-network computations faster and more efficient, especially for large datasets and GPU-based computation.

10. What is the purpose of an activation function?

Answer:
The main purpose of an activation function is to introduce **non-linearity** into a neural network.

Without nonlinear activation functions, multiple layers would behave like a single linear transformation and the network would have limited ability to learn complex patterns.

Common activation functions include:

- Sigmoid
- Tanh

- ReLU

11. Define a shallow neural network.

Answer:
A shallow neural network is a neural network that generally contains **one hidden layer** between the input and output layers.

Its basic structure is:

Input Layer→Hidden Layer→Output Layer

It can be used for classification and regression problems.

12. What is a deep neural network?

Answer:
A deep neural network (DNN) is a neural network containing **multiple hidden layers** between the input and output layers.

Its general structure is:

$$\text{Input} \rightarrow \text{Hidden Layer}_1 \rightarrow \text{Hidden Layer}_2 \rightarrow \cdots \rightarrow \text{Output}$$

Deep neural networks can learn complex patterns and hierarchical representations from data.

13. Why is ReLU widely used in deep learning?

Answer:
ReLU (Rectified Linear Unit) is widely used because it is computationally simple and introduces non-linearity.

It is defined as:

$$ReLU(z) = \max(0, z)$$

Advantages include:

- Simple and fast computation
- Helps reduce the vanishing-gradient problem compared with sigmoid and tanh in many deep networks
- Efficient optimization in many practical applications

14. What is the learning rate?

Answer:
The **learning rate**, represented by α\alpha, determines the size of the step taken when updating the parameters of a neural network during optimization.

For gradient descent:

$$W := W - \alpha \frac{\partial J}{\partial W}$$

If the learning rate is too small, training can be very slow. If it is too large, training may become unstable or fail to converge.

15. What is gradient descent?

Answer:
Gradient descent is an optimization algorithm used to **minimize the loss or cost function** of a machine-learning model.

The parameters are updated in the opposite direction of the gradient:

$$W := W - \alpha \frac{\partial J}{\partial W}$$
$$b := b - \alpha \frac{\partial J}{\partial b}$$

The process is repeated until the loss is sufficiently minimized or a stopping criterion is reached.

Long-Answer Questions

1. Explain the architecture and working of a neural network.

A **neural network** is a computational model inspired by the working of the human brain. It consists of interconnected processing units called **neurons**, arranged in layers. Neural networks are widely used for classification, regression, pattern recognition, image processing, natural language processing, and many other deep learning applications.

Architecture of a Neural Network

A basic neural network consists of:

1. **Input Layer**
2. **One or more Hidden Layers**
3. **Output Layer**

A neuron receives inputs, multiplies them by corresponding weights, adds a bias, and applies an activation function.

For a neuron with inputs $x_1, x_2, \ldots, x_n$,

$$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

or

$$z = \sum_{i=1}^{n} w_ix_i + b$$

The output of the neuron is

$$a = g(z)$$

where g is the activation function.

Working

The working of a neural network can be summarized as:

Input → Weighted Sum → Activation Function → Hidden Layers → Output

During **forward propagation**, input data passes through the network from the input layer toward the output layer.

The predicted output is compared with the actual output using a **loss function**. The error is then propagated backward through the network using **backpropagation**.

The weights and biases are updated using an optimization algorithm such as **gradient descent**.

Example

For a binary classification problem, the output neuron may use the sigmoid function:

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

The output lies between 0 and 1 and can be interpreted as the probability of belonging to the positive class.

Conclusion

Thus, a neural network learns a mapping between input and output by adjusting its weights and biases during training.

2. Explain binary classification using logistic regression.

Logistic regression is a supervised machine learning algorithm used mainly for **binary classification**, where the output belongs to one of two classes, generally represented by 0 and 1.

Examples include:

- Disease: Yes/No
- Spam: Yes/No
- Pass/Fail
- Customer churn: Yes/No

Model

For an input vector x, logistic regression first computes a linear combination:

$$z = w^T x + b$$

where:

- w = weight vector
- x = input vector
- b = bias
- z = linear score

The sigmoid function is then applied:

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Thus,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x + b)}}$$

The output $\hat{y}$ represents the estimated probability that the input belongs to class 1.

Classification Rule

A threshold is normally selected as 0.5.

$$\hat{y} \geq 0.5 \Rightarrow \text{Class } 1$$

$$\hat{y} < 0.5 \Rightarrow \text{Class } 0$$

Cost Function

For binary classification, the **binary cross-entropy loss** is commonly used:

$$L(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

For m training examples,

$$J(w, b) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

Training

The objective is to minimize $J(w, b)$.

Gradient descent is used:

$$w := w - \alpha \frac{\partial J}{\partial w}$$

$$b := b - \alpha \frac{\partial J}{\partial b}$$

where α is the learning rate.

Conclusion

Logistic regression provides a simple foundation for understanding neural networks because a single logistic-regression neuron performs a similar computation to a basic neural-network output neuron.

3. Explain the sigmoid activation function with its derivative.

The **sigmoid function** is one of the important activation functions used in neural networks, particularly in binary classification problems.

Definition

The sigmoid function is given by

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

Its output lies between 0 and 1:

$$0 < \sigma(z) < 1$$

Therefore, it can be interpreted as a probability.

Properties

1. Output range is (0,1)(0,1).
2. It is differentiable.
3. It is nonlinear.
4. It is suitable for binary classification.
5. It is monotonically increasing.
6. It approaches 0 for large negative z.
7. It approaches 1 for large positive z.

Derivation of the Derivative

Starting with

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

Rewrite as

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

Differentiating,

$$\frac{d\sigma}{dz} = -(1 + e^{-z})^{-2}(-e^{-z})$$

Therefore,

$$\frac{d\sigma}{dz} = \frac{e^{-z}}{(1 + e^{-z})^2}$$

This can be expressed in terms of sigmoid itself:

$$\boxed{\sigma'(z) = \sigma(z)[1 - \sigma(z)]}$$

Importance of the Derivative

The derivative is required during **backpropagation** to calculate gradients and update neural-network parameters.

Limitation

For very large positive or negative values of z, the sigmoid derivative becomes very small. This can result in the **vanishing-gradient problem**.

Therefore, sigmoid is generally preferred at the output layer for binary classification, while ReLU-type functions are commonly used in hidden layers.

4. Explain computation graphs and their role in deep learning.

A **computation graph** is a graphical representation of mathematical operations used to calculate an output from input variables.

It represents:

- Variables as nodes
- Mathematical operations as edges or operation nodes

Example

Consider:

$$z = wx + b$$

$$a = \sigma(z)$$

The computation can be represented as:

$x, w \rightarrow$ **multiplication** $\rightarrow wx \rightarrow$ **addition with** $b \rightarrow z \rightarrow$ **sigmoid** $\rightarrow a$

For a loss function,

$$L(a, y)$$

the complete computation becomes:

$$x, w, b \rightarrow z \rightarrow a \rightarrow L$$

Forward Computation

During forward propagation, values are calculated from the input toward the output.

For example:

$$z = wx + b$$

$$a = \sigma(z)$$

$$L = L(a, y)$$

Backward Computation

During backpropagation, derivatives are calculated from the loss backward toward the parameters.

Using the chain rule,

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w}$$

Role in Deep Learning

Computation graphs are important because they:

1. Represent complex mathematical models.
2. Make forward propagation systematic.
3. Enable automatic differentiation.
4. Simplify gradient computation.
5. Support backpropagation.
6. Help implement deep neural networks efficiently.
7. Allow modern deep-learning frameworks to automatically calculate gradients.

Conclusion

A computation graph provides the mathematical structure required for **forward computation and backward gradient computation**, which are fundamental to neural-network training.

5. Explain forward propagation and backpropagation.

Forward propagation and **backpropagation** are two fundamental processes used to train a neural network.

Forward Propagation

In forward propagation, input data moves from the input layer toward the output layer.

For layer l,

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$

and

$$A^{[l]} = g^{[l]}(Z^{[l]})$$

where:

- $W^{[l]}$ = weight matrix
- $b^{[l]}$ = bias vector
- $A^{[l-1]}$ = previous layer activation
- $Z^{[l]}$ = weighted input
- $A^{[l]}$ = activation

At the output layer, the network produces the prediction:

$$\hat{y} = A^{[L]}$$

The prediction is compared with the actual target y using a loss function.

Backpropagation

Backpropagation calculates how much each parameter contributed to the error.

It uses the **chain rule of differentiation** to propagate the error from the output layer toward the input layer.

For a parameter w,

$$\frac{\partial L}{\partial w}$$

is calculated.

The parameters are then updated using gradient descent:

$$w := w - \alpha \frac{\partial L}{\partial w}$$

$$b := b - \alpha \frac{\partial L}{\partial b}$$

Complete Process

Input→ Forward Propagation→ Prediction→ Loss→Backpropagation→Parameter Update

This process is repeated for many iterations or epochs.

Conclusion

Forward propagation produces predictions, while backpropagation calculates gradients required to improve those predictions.

6. Explain vectorization in logistic regression.

Vectorization is the process of performing mathematical operations on entire vectors or matrices rather than processing individual elements using explicit loops.

It is extremely important in deep learning because it makes computations faster and more efficient.

Non-vectorized Approach

Suppose there are m training examples. The prediction for each example can be calculated separately:

$$z^{(i)} = w^T x^{(i)} + b$$

for

$$i = 1, 2, \ldots, m$$

This approach requires repeated computations.

Vectorized Approach

Let

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

Then all predictions can be computed together:

$$Z = w^T X + b$$

The sigmoid is applied element-wise:

$$A = \sigma(Z)$$

Thus, a large number of examples can be processed simultaneously.

Gradient Computation

For logistic regression,

$$dZ = A - Y$$

The gradient with respect to weights is

$$dw = \frac{1}{m} X \, dZ^T$$

and the gradient with respect to bias is

$$db = \frac{1}{m}\sum_{i=1}^{m} dZ^{(i)}$$

Advantages

Vectorization:

1. Reduces execution time.
2. Uses matrix operations efficiently.
3. Avoids unnecessary loops.
4. Makes implementation simpler.
5. Allows efficient use of CPUs and GPUs.
6. Is essential for large-scale deep-learning models.

Conclusion

Vectorization allows logistic regression and neural networks to process many training examples simultaneously, significantly improving computational efficiency.

7. Explain the architecture of a shallow neural network.

A **shallow neural network** generally consists of an input layer, one hidden layer, and an output layer.

Architecture

The basic structure is:

Input Layer→Hidden Layer→Output Layer

Suppose the input vector is

$$X = [x_1, x_2, \ldots, x_n]^T$$

For the hidden layer,

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = g^{[1]}(Z^{[1]})$$

For the output layer,

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = g^{[2]}(Z^{[2]})$$

The output $A^{[2]}$ represents the prediction.

Components

1. Input Layer

Receives the input features.

2. Hidden Layer

Performs nonlinear transformations using weights, biases, and activation functions.

3. Output Layer

Produces the final prediction.

Example

For binary classification:

- Hidden layer: ReLU or another nonlinear activation
- Output layer: sigmoid

$$\hat{y} = \sigma(W^{[2]}A^{[1]} + b^{[2]})$$

Advantages

- More powerful than logistic regression.
- Can learn nonlinear relationships.
- Simple compared with very deep networks.

Limitation

A shallow network may have difficulty learning highly complex hierarchical features that can be learned using deeper architectures.

8. Explain the building blocks of a deep neural network.

A **Deep Neural Network (DNN)** consists of multiple interconnected layers that transform input data into useful representations and finally produce an output.

Major Building Blocks

1. Input Layer

Receives the input features.

2. Hidden Layers

A DNN contains multiple hidden layers.

Each layer performs:

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$

followed by

$$A^{[l]} = g^{[l]}(Z^{[l]})$$

3. Weights

Weights determine the importance of input signals.

They are learned during training.

4. Bias

Bias provides an additional adjustable parameter:

$$Z = WX + b$$

5. Activation Functions

Activation functions introduce nonlinearity.

Common functions include:

- Sigmoid
- Tanh
- ReLU
- Leaky ReLU

- Softmax

6. Loss Function

Measures the difference between predicted and actual outputs.

Examples include:

- Mean Squared Error
- Binary Cross-Entropy
- Categorical Cross-Entropy

7. Optimizer

Updates the weights to minimize the loss.

Examples:

- Gradient Descent
-
- Stochastic Gradient Descent
- Adam
- RMSprop

8. Backpropagation

Calculates gradients of the loss with respect to network parameters.

9. Learning Rate

Controls the size of parameter updates.

Overall Structure

$$X \to [W^{[1]}, b^{[1]}, g^{[1]}] \to [W^{[2]}, b^{[2]}, g^{[2]}] \to \cdots \to \hat{Y}$$

Conclusion

The combination of layers, weights, biases, activation functions, loss functions, backpropagation, and optimization enables a DNN to learn complex patterns.

9. Derive the basic equations used in forward propagation.

Consider a neural network with L layers.

Let:

- $A^{[0]} = X$
- $W^{[l]}$ = weight matrix of layer l
- $b^{[l]}$ = bias vector
- $Z^{[l]}$ = weighted input
- $A^{[l]}$ = activation
- $g^{[l]}$ = activation function

Step 1: Input Layer

The input is represented as

$$A^{[0]} = X$$

Step 2: First Hidden Layer

The weighted sum is

$$Z^{[1]} = W^{[1]}A^{[0]} + b^{[1]}$$

Applying the activation function,

$$A^{[1]} = g^{[1]}(Z^{[1]})$$

Step 3: Second Layer

Similarly,

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

and

$$A^{[2]} = g^{[2]}(Z^{[2]})$$

Step 4: General Layer

For any layer l,

$$\boxed{Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}}$$

and

$$\boxed{A^{[l]} = g^{[l]}(Z^{[l]})}$$

Output Layer

For the final layer L,

$$Z^{[L]} = W^{[L]}A^{[L-1]} + b^{[L]}$$

$$\hat{Y} = A^{[L]} = g^{[L]}(Z^{[L]})$$

The complete forward propagation is therefore:

$$A^{[0]} = X$$
$$Z^{[1]} = W^{[1]} A^{[0]} + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$\vdots$$
$$Z^{[L]} = W^{[L]} A^{[L-1]} + b^{[L]}$$
$$\hat{Y} = A^{[L]} = g^{[L]}(Z^{[L]})$$

These equations form the mathematical foundation of forward propagation in deep neural networks.

10. Explain the complete training process of a neural network.

Training a neural network is an iterative process in which the network learns appropriate weights and biases from training data.

Step 1: Collect and Prepare Data

The dataset is collected and divided into:

- Training data
- Validation data
- Test data

Data may also be normalized or standardized.

Step 2: Initialize Parameters

Weights and biases are initialized.

$W^{[l]}, b^{[l]}$ are assigned initial values.

Weights are generally initialized using suitable initialization methods rather than all being set to zero.

Step 3: Forward Propagation

The input passes through all network layers.

For every layer,

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$

$$A^{[l]} = g^{[l]}(Z^{[l]})$$

The final output is the prediction:

Step 4: Calculate Loss

The predicted output is compared with the actual output.

For binary classification:

$$J = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

A lower loss indicates better performance on the training data.

Step 5: Backpropagation

The gradient of the loss with respect to every parameter is calculated.

Using the chain rule:

$$\frac{\partial J}{\partial W^{[l]}}$$

and

$$\frac{\partial J}{\partial b^{[l]}}$$

are determined for all layers.

Step 6: Update Parameters

Using gradient descent:

$$W^{[l]} = W^{[l]} - \alpha \frac{\partial J}{\partial W^{[l]}}$$

$$b^{[l]} = b^{[l]} - \alpha \frac{\partial J}{\partial b^{[l]}}$$

where α is the learning rate.

Step 7: Repeat

The forward propagation, loss calculation, backpropagation, and parameter update steps are repeated for multiple batches and epochs.

Step 8: Validation

The model is evaluated on validation data to monitor:

- Generalization
- Overfitting
- Hyperparameter performance

Techniques such as **dropout, regularization, early stopping, and batch normalization** may be used to improve training and generalization.

Step 9: Testing

After training is complete, the final model is evaluated using unseen test data.

Common metrics include:

- Accuracy
- Precision
- Recall
- F1-score
- Mean Squared Error, depending on the task

Complete Training Cycle

Data Preparation → Parameter Initialization → Forward Propagation → Prediction → Loss Calculation → Backpropagation → Gradient Calculation → Parameter Update → Repeat for Multiple Epochs → Validation → Testing → Final Trained Model

The complete training process enables a neural network to automatically learn the relationship between inputs and outputs by minimizing a loss function through repeated forward and backward computations.

Module 2: Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization — 24 Hours

Sub-Topics:

- Practical Aspects of Deep Learning
- Understanding Exponentially Weighted Averages
- Weight Initialization for Deep Networks
- The Problem of Local Optima
- Optimization Algorithms
- Hyperparameter Tuning, Batch Normalization and Programming Frameworks

Formative Assessment:

- 3 Quizzes
- 5 Coding/Lab Assignments

1. Practical Aspects of Deep Learning

1.1 Introduction

Building a deep neural network is not only about designing the network architecture. A successful deep-learning model requires appropriate decisions regarding:

- Training, validation and test datasets
- Bias and variance
- Regularization
- Initialization
- Optimization
- Hyperparameter tuning
- Batch normalization
- Programming frameworks

The objective is to develop a model that performs well not only on training data but also on **unseen data**.

1.2 Training, Validation and Test Sets

A dataset is commonly divided into:

Training Set

Used to train the neural network and learn:

$$W, b$$

Validation/Development Set

Used to:

- Select the best model
- Tune hyperparameters
- Compare different architectures
- Detect overfitting

Test Set

Used only for the **final evaluation** of the selected model.

A typical division may be:

$$\text{Dataset} \rightarrow \begin{cases} \text{Training Set} \\ \text{Validation Set} \\ \text{Test Set} \end{cases}$$

The exact percentages depend on the size and characteristics of the dataset.

2. Bias and Variance

Two important sources of prediction error are **bias** and **variance**.

High Bias

A model with high bias is generally **underfitting** the data.

Symptoms:

- High training error
- High validation error

Example:

$$J_{train} = 15\%, \qquad J_{dev} = 16\%$$

The model is performing poorly even on the training data.

High Variance

A model with high variance is generally **overfitting** the training data.

Symptoms:

- Very low training error
- High validation error

Example:

$$J_{train} = 2\%, \qquad J_{dev} = 15\%$$

Desired Situation

A good model should have:

- Low training error
- Low validation error
- Small difference between training and validation performance

3. Regularization

Regularization is a technique used to reduce overfitting.

It adds a penalty to the cost function so that the model does not rely excessively on large weights.

Common regularization methods include:

1. L2 regularization
2. L1 regularization
3. Dropout
4. Data augmentation
5. Early stopping

For this course, **L2 regularization and dropout** are particularly important.

4. L2 Regularization

In L2 regularization, a penalty proportional to the squared magnitude of the weights is added to the cost function.

The regularized cost function is:

$$J_{reg} = J + \frac{\lambda}{2m} \sum_{l} \|W^{[l]}\|_F^2$$

Where :

- J = original cost
- λ = regularization parameter
- m = number of training examples
- $W^{[l]}$ = weight matrix of layer l

The Frobenius norm is:

$$\|W\|_F^2 = \sum_i \sum_j W_{ij}^2$$

Effect

A large value of λ\lambda strongly penalizes large weights.

The objective becomes:

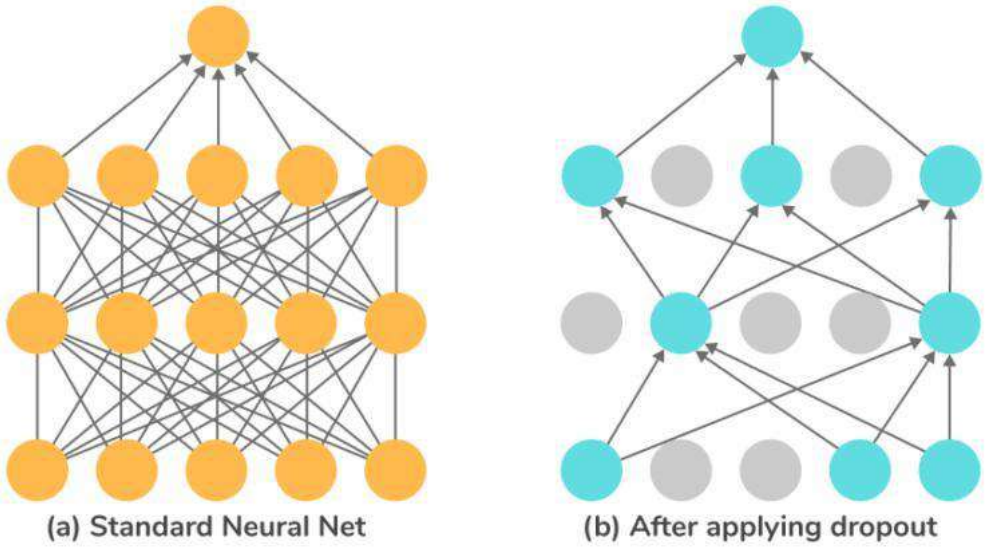
Minimize prediction error + complexity penalty\text{Minimize prediction error + complexity penalty}

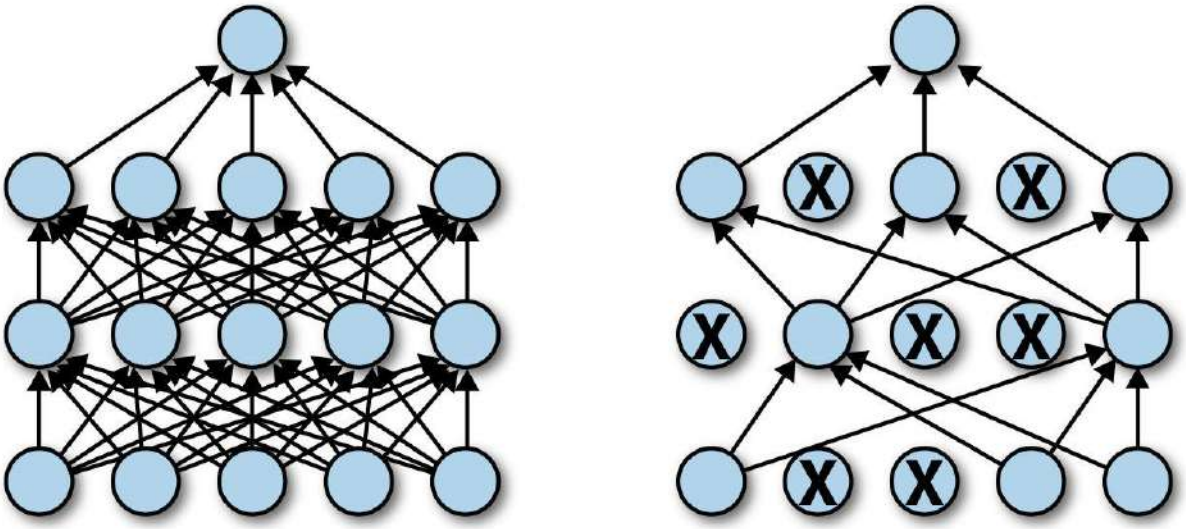
Important Point

L2 regularization generally encourages the weights to become smaller rather than exactly zero.

5. Dropout Regularization

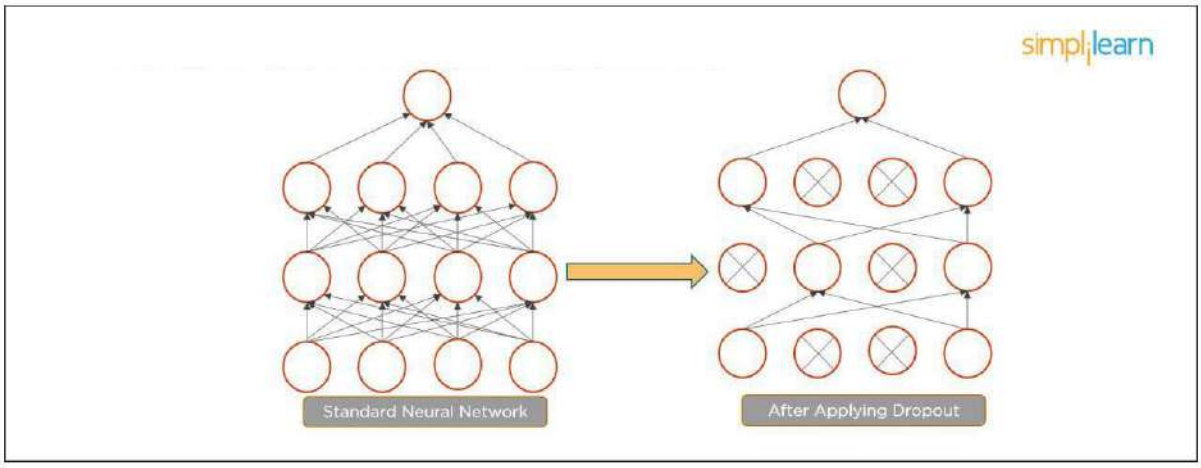
Dropout is a regularization technique in which some neurons are randomly ignored during training.

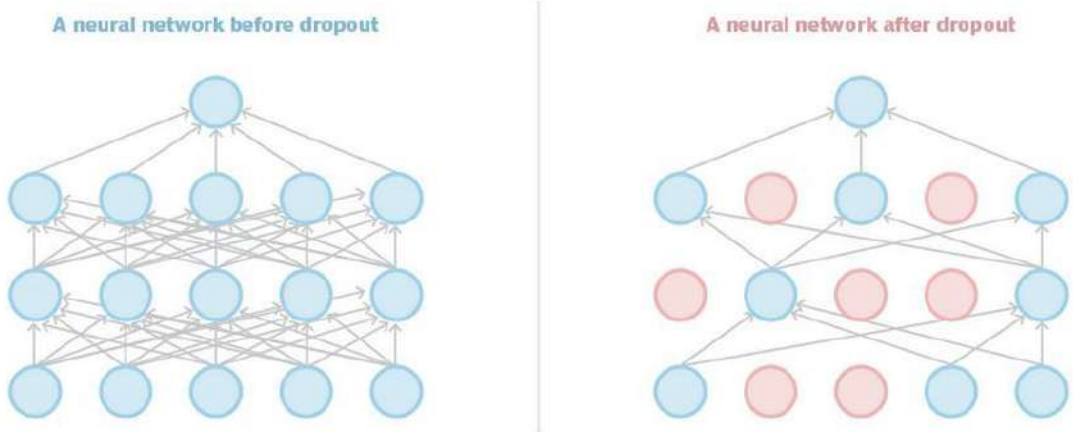




(a) Standard Neural Net

(b) After applying dropout





Suppose the activation of a layer is:

$$A^{[l]}$$

A dropout mask $D^{[l]}$ is generated:

$$D^{[l]} \sim Bernoulli(p)$$

The activations are modified:

$$\tilde{A}^{[l]} = D^{[l]} \odot A^{[l]}$$

With inverted dropout, the activations are scaled:

$$\tilde{A}^{[l]} = \frac{D^{[l]} \odot A^{[l]}}{p}$$

where p is the keep probability.

Advantages

- Reduces overfitting
- Prevents excessive dependence on particular neurons
- Encourages distributed representations

Important Rule

Dropout is generally used during training, not during inference/testing.

6. Understanding Exponentially Weighted Averages

Exponentially weighted averages are important in optimization algorithms such as **Momentum** and **RMSprop**.

Suppose we have a sequence:

$$\theta_1, \theta_2, \theta_3, \ldots$$

An exponentially weighted average can be written as:

$$V_t = \beta V_{t-1} + (1 - \beta)\theta_t$$

where:

- V_t = current weighted average
- β = weighting parameter
- θ_t = current value

For example, if:

$$\beta = 0.9$$

then:

$$V_t = 0.9V_{t-1} + 0.1\theta_t$$

Recent values receive greater importance, while older values gradually receive less influence.

7. Bias Correction

At the beginning of training, the exponentially weighted average may be biased toward zero because:

$$V_0 = 0$$

A bias-corrected estimate can be obtained using:

$$\hat{V}_t = \frac{V_t}{1 - \beta^t}$$

This concept is especially important in **Adam optimization**.

8. Weight Initialization for Deep Networks

The initial values of weights have a major influence on neural-network training.

Poor initialization can result in:

- Vanishing gradients
- Exploding gradients
- Slow convergence
- Poor learning

Therefore, weights should generally not be initialized to the same nonzero value.

8.1 Why Not Initialize All Weights to Zero?

Suppose all neurons in a layer have:

$$W = 0$$

Then all neurons receive identical gradients and remain identical.

This is called a **symmetry problem**.

Therefore, random or carefully designed initialization is preferred.

9. Xavier/Glorot Initialization

Xavier initialization is designed to maintain a suitable variance of activations across layers.

A common form is:

$$W_{ij} \sim N\left(0, \frac{1}{n_{in}}\right)$$

or, in a symmetric uniform formulation, the variance is chosen based on both input and output dimensions.

It is particularly associated with sigmoid/tanh-type activations.

10. He Initialization

He initialization is particularly suitable for ReLU-based networks.

A common formulation is:

$$W_{ij} \sim N\left(0, \frac{2}{n_{in}}\right)$$

where n_{in} is the number of input units to the layer.

Advantage

It helps maintain the scale of activations and gradients when ReLU activations are used.

11. The Problem of Local Optima

Optimization aims to minimize a cost function:

$$J(W, b)$$

A **local minimum** is a point where the cost is lower than nearby points but may not be the lowest possible value globally.

In modern high-dimensional deep-learning problems, the optimization landscape can contain:

- Local minima
- Saddle points
- Flat regions
- Steep regions

Local Minimum

$$J(\theta^*) < J(\theta)$$

for nearby values of θ.

Saddle Point

A saddle point is a point where the gradient may be zero, but the point is neither a local maximum nor a local minimum.

In practice, **saddle points and poorly conditioned regions** can be more important optimization challenges than simple local minima.

12. Gradient Descent

Gradient descent minimizes a cost function by moving parameters in the direction opposite to the gradient.

For parameter θ\theta:

$$\theta_t = \theta_{t-1} - \alpha \frac{\partial J}{\partial \theta}$$

where α is the learning rate.

Batch Gradient Descent

Uses the entire training dataset to calculate one gradient update.

Advantage: Stable gradient estimate.

Disadvantage: Computationally expensive for large datasets.

13. Mini-Batch Gradient Descent

Instead of using the entire dataset, the training data is divided into smaller batches.

For example:

$$m = 10,000$$

training examples may be divided into batches of: 64, 128, 256

examples.

Each mini-batch produces one parameter update.

Advantages

- Faster than full-batch gradient descent
- More efficient for large datasets
- Works well with GPUs
- Provides frequent parameter updates

14. Momentum Optimization

Momentum uses an exponentially weighted average of past gradients. Let,

$$V_{dW} = \beta V_{dW} + (1-\beta)dW$$

$$V_{db} = \beta V_{db} + (1-\beta)db$$

Then:

$$W := W - \alpha V_{dW}$$

$$b := b - \alpha V_{db}$$

Advantages

- Reduces oscillations
- Accelerates movement in useful directions

- Often converges faster than ordinary gradient descent

15. RMSprop

RMSprop maintains an exponentially weighted average of squared gradients.

$$S_{dW} = \beta S_{dW} + (1-\beta)(dW)^2$$

The parameter update is:

$$W = W - \alpha \frac{dW}{\sqrt{S_{dW} + \epsilon}}$$

where ϵ is a small positive number used to prevent division by zero.

Advantages

- Adaptive learning rate
- Helps handle different gradient scales
- Can improve convergence

16. Adam Optimization: **Adam** stands for **Adaptive Moment Estimation**. It combines ideas from Momentum and RMSprop.

First moment:

$$V_t = \beta_1 V_{t-1} + (1-\beta_1) g_t$$

Second moment:

$$S_t = \beta_2 S_{t-1} + (1-\beta_2) g_t^2$$

Bias correction:

$$\hat{V}_t = \frac{V_t}{1-\beta_1^t}$$

$$\hat{S}_t = \frac{S_t}{1-\beta_2^t}$$

Parameter update:

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{V}_t}{\sqrt{\hat{S}_t} + \epsilon}$$

17. Comparison of Optimization Algorithms:

Algorithm	Main Idea	Major Advantage
Batch GD	Uses entire dataset	Stable gradient
Mini-batch GD	Uses small batches	Efficient
Momentum	Uses moving average of gradients	Reduces oscillations
RMSprop	Uses moving average of squared gradients	Adaptive step sizes
Adam	Combines momentum and RMSprop ideas	Fast and adaptive

18. Hyperparameter Tuning

Hyperparameters are settings chosen before or during training rather than learned directly as network weights.

Important hyperparameters include:

- Learning rate
- Number of hidden layers
- Number of neurons
- Mini-batch size
- Number of epochs
- Regularization parameter λ
- Dropout keep probability
- Optimization algorithm
- Activation function

19. Learning Rate as an Important Hyperparameter

The learning rate α is one of the most important hyperparameters.

Too Small: $\alpha \rightarrow$very small

Training becomes slow.

Too Large: $\alpha \rightarrow$very large

The algorithm may overshoot the minimum and fail to converge.

Suitable Learning Rate

A suitable value allows the cost function to decrease steadily.

20. Hyperparameter Search Methods

Common methods include:

20.1 Grid Search

Different combinations of predefined hyperparameter values are tested.

Example:

$$\alpha \in \{0.001, 0.01, 0.1\}$$
$$\lambda \in \{0.001, 0.01, 0.1\}$$

Every combination is evaluated.

20.2 Random Search

Random combinations of hyperparameters are selected.

It can be more efficient than grid search when only a few hyperparameters strongly influence performance.

20.3 Coarse-to-Fine Search

A broad range is searched first. After identifying a promising region, a finer search is performed around that region.

21. Batch Normalization

Batch normalization normalizes the activations of a neural network layer.

For a mini-batch:

$$\mu_B = \frac{1}{m_B} \sum_{i=1}^{m_B} x_i$$

and

$$\sigma_B^2 = \frac{1}{m_B} \sum_{i=1}^{m_B} (x_i - \mu_B)^2$$

Normalize:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Then scale and shift:

$$y_i = \gamma \hat{x}_i + \beta$$

where:

- γ = learnable scaling parameter
- β = learnable shifting parameter

Advantages

- Can improve training stability
- Often allows faster training
- Helps manage activation distributions
- Can provide some regularization effect

22. Batch Normalization During Training and Inference

During training, batch statistics are used.

During inference, the model generally uses **running estimates** of the mean and variance accumulated during training.

This ensures stable predictions even when the inference batch is small or consists of a single example.

23. Programming Frameworks

Deep-learning frameworks provide tools for constructing, training, evaluating, and deploying neural networks.

Important frameworks include:

TensorFlow

Developed as a comprehensive machine-learning framework supporting:

- Tensor operations
- Automatic differentiation
- Neural-network construction
- GPU/accelerator computation
- Model training and deployment

Keras

Keras provides a high-level API for constructing neural networks with a relatively simple programming interface.

Typical workflow:

```
model = Sequential()

model.add(Dense(64, activation='relu'))

model.add(Dense(1, activation='sigmoid'))
```

The exact API may vary by framework/version.

PyTorch

PyTorch is another widely used deep-learning framework supporting:

- Tensor computation
- Automatic differentiation
- Neural networks
- GPU acceleration
- Research and production workflows

24. TensorFlow Neural Network Workflow

A typical TensorFlow/Keras workflow is:

Step 1: Import libraries

```
import tensorflow as tf
```

Step 2: Create the model

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Step 3: Compile the model:

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

Step 4: Train

```
model.fit(X_train, y_train, epochs=20, batch_size=32)
```

Step 5: Evaluate

```
model.evaluate(X_test, y_test)
```

25. Practical Deep Learning Workflow

A good deep-learning project can be organized as:

Data → Split Data → Base Line Model → Diagnose Bias/ Variance → Regularization
Optimization → Hyperparameter Tuning → Validation → Final Testing →

26. Important Concepts for Examination

Regularization

Used primarily to reduce: Overfitting

L2 Regularization

Adds:

$$\frac{\lambda}{2m}\sum_{l}\|W^{[l]}\|_F^2$$

to the cost function.

Dropout

Randomly removes neurons during training.

Momentum

Uses an exponentially weighted average of gradients.

RMSprop

Uses an exponentially weighted average of squared gradients.

Adam

Combines first-moment and second-moment estimates with bias correction.

Batch Normalization

Normalizes layer activations and then applies learnable scale and shift.

Hyperparameter Tuning

Finds suitable values of parameters such as learning rate, batch size, number of layers, regularization strength, and dropout probability.

Lab Assignments

1. **Implement Mini-Batch Gradient Descent** for a simple neural network.
2. **Implement L2 Regularization** and compare training/validation performance.
3. **Implement Dropout** and analyze its effect on overfitting.
4. **Compare Momentum, RMSprop and Adam** on the same dataset.
5. **Build and train a neural network using TensorFlow/Keras**, including batch normalization and hyperparameter tuning.

Short Questions with Answers

Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization

Q1. What is the main objective of deep learning?

Answer: To develop a model that learns patterns from data and performs well on unseen data.

Q2. What is a training set?

Answer: A training set is used to train a neural network and learn its weights and biases.

Q3. What is a validation set?

Answer: A validation set is used to select models, tune hyperparameters and detect overfitting.

Q4. What is a test set?

Answer: A test set is used for the final evaluation of the selected model.

Q5. What is bias?

Answer: Bias is the error associated with a model's inability to learn the underlying patterns adequately.

Q6. What is variance?

Answer: Variance refers to a model's sensitivity to variations in training data, often resulting in overfitting.

Q7. What is underfitting?

Answer: Underfitting occurs when a model fails to learn the training data adequately.

Q8. What is overfitting?

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

B. Regularization

Q9. What is regularization?

Answer: Regularization is a technique used to reduce overfitting by penalising model complexity or modifying training.

Q10. Name five regularization techniques.

Answer: L1 regularization, L2 regularization, dropout, data augmentation and early stopping.

Q11. What is L1 regularization?

Answer: L1 regularization adds a penalty proportional to the absolute values of model weights to the cost function.

Q12. What is L2 regularization?

Answer: L2 regularization adds a penalty proportional to the squared values of model weights to the cost function.

Q13. What is the purpose of the regularization parameter λ?

Answer: It controls the strength of the regularization penalty.

Q14. What is dropout?

Answer: Dropout is a regularization technique in which some neurons are randomly deactivated during training.

Q15. State two advantages of dropout.

Answer: It reduces overfitting and prevents excessive dependence on particular neurons.

Q16. When is dropout generally used?

Answer: Dropout is generally used during training and disabled during inference.

C. Exponentially Weighted Averages and Initialization

Q17. What is an exponentially weighted average?

Answer: It is a weighted average in which recent values receive greater importance than older values.

Q18. Name two optimization algorithms that use exponentially weighted averages.

Answer: Momentum and RMSprop.

Q19. What is bias correction in optimization?

Answer: Bias correction compensates for the initial bias toward zero in exponentially weighted estimates.

Q20. Why is weight initialization important?

Answer: Proper weight initialization helps improve learning and convergence and reduces the risk of vanishing or exploding gradients.

Q21. Why should all neural-network weights not be initialized to the same value?

Answer: Identical initialization can cause neurons to learn identical features because they receive identical gradients, creating a symmetry problem.

Q22. What is Xavier initialization?

Answer: Xavier or Glorot initialization aims to maintain a suitable variance of activations across layers and is commonly associated with sigmoid and tanh activations.

Q23. What is He initialization?

Answer: He initialization is designed particularly for ReLU-based neural networks to help maintain the scale of activations and gradients.

D. Optimization Algorithms

Q24. What is a local minimum?

Answer: A local minimum is a point where the cost is lower than at nearby points but may not be the lowest possible cost globally.

Q25. What is a saddle point?

Answer: A saddle point is a point where the gradient may be zero, but the point is neither a local minimum nor a local maximum.

Q26. What is gradient descent?

Answer: Gradient descent minimises a cost function by updating parameters in the direction opposite to the gradient.

Q27. What is the learning rate?

Answer: The learning rate determines the step size used to update model parameters during training.

Q28. What is batch gradient descent?

Answer: Batch gradient descent calculates each gradient update using the entire training dataset.

Q29. What is mini-batch gradient descent?

Answer: Mini-batch gradient descent divides training data into small batches and updates the parameters using each batch.

Q30. State two advantages of mini-batch gradient descent.

Answer: It provides frequent parameter updates and is computationally efficient for large datasets.

Q31. What is momentum optimization?

Answer: Momentum uses an exponentially weighted average of past gradients to reduce oscillations and accelerate convergence.

Q32. What is RMSprop?

Answer: RMSprop uses an exponentially weighted average of squared gradients to adapt the effective learning rate.

Q33. What is Adam optimization?

Answer: Adam stands for Adaptive Moment Estimation. It combines ideas from Momentum and RMSprop and uses bias correction.

Q34. State one advantage of Adam.

Answer: Adam provides adaptive parameter updates and often converges efficiently.

E. Hyperparameter Tuning

Q35. What is a hyperparameter?

Answer: A hyperparameter is a setting selected before or during training that is not directly learned as a model weight.

Q36. Name five important hyperparameters.

Answer: Learning rate, batch size, number of epochs, number of hidden layers and regularization strength.

Q37. What happens when the learning rate is too small?

Answer: Training becomes slow because the parameter updates are very small.

Q38. What happens when the learning rate is too large?

Answer: The optimisation process may overshoot the minimum or fail to converge.

Q39. What is grid search?

Answer: Grid search tests predefined combinations of hyperparameter values to identify a suitable combination.

Q40. What is random search?

Answer: Random search selects and evaluates hyperparameter combinations randomly from specified ranges or distributions.

Q41. What is coarse-to-fine search?

Answer: It first searches a broad range of hyperparameters and then performs a more detailed search around promising values.

F. Batch Normalization and Programming Frameworks

Q42. What is batch normalization?

Answer: Batch normalization normalizes layer activations and applies learnable scale and shift parameters.

Q43. State two advantages of batch normalization.

Answer: It can improve training stability and often allows faster training.

Q44. How is batch normalization performed during inference?

Answer: It generally uses running estimates of the mean and variance accumulated during training.

Q45. What is TensorFlow?

Answer: TensorFlow is a machine-learning framework that supports tensor operations, automatic differentiation, neural-network training and deployment.

Q46. What is Keras?

Answer: Keras is a high-level API for building and training neural networks using a relatively simple programming interface.

Q47. What is PyTorch?

Answer: PyTorch is a deep-learning framework that supports tensor computation, automatic differentiation, neural networks and GPU acceleration.

Q48. What is the purpose of the Adam optimizer in TensorFlow/Keras?

Answer: Adam adaptively updates neural-network parameters using estimates of the first and second moments of gradients.

Q49. What is an epoch?

Answer: An epoch is one complete pass through the entire training dataset.

Q50. What is the purpose of model evaluation?

Answer: Model evaluation measures how well the trained model performs on data not used to fit its parameters.

Important formulas for quick revision

Concept	Formula
L2 regularization	$J_{\text{reg}} = J + \frac{\lambda}{2m} \sum_l \|W^{[l]}\|_F^2$
Gradient descent	$\theta := \theta - \alpha \nabla_\theta J(\theta)$
Exponentially weighted average	$v_t = \beta v_{t-1} + (1-\beta)\theta_t$, where θ_t denotes the current sequence value
Momentum	$v_t = \beta v_{t-1} + (1-\beta) g_t$
RMSprop	$s_t = \beta s_{t-1} + (1-\beta) g_t^2$
Adam	Combines first- and second-moment estimates with bias correction

Module 3: Structuring Machine Learning Projects [7 Hours]

Sub-Topic

Why ML Strategy

Understanding Human-level Performance

Surpassing Human-level Performance

Transfer Learning

Multi-task Learning

What is End-to-end Deep Learning?

Module 3: Structuring Machine Learning Projects

1. Introduction to Structuring Machine Learning Projects

Machine Learning (ML) projects require more than selecting an algorithm and training a model. A successful project involves defining the objective, selecting suitable data, choosing evaluation metrics, analysing errors, and deciding which improvements will provide the greatest benefit.

Structuring a Machine Learning project means developing a systematic strategy to build, evaluate, improve, and deploy an ML system efficiently.

For example, consider an image-classification system designed to detect defective electrical components. Its development involves:

1. Defining the classification problem.
2. Collecting and labelling images.
3. Establishing human-level or other baseline performance.
4. Training and evaluating the model.
5. Analysing errors and identifying bottlenecks.
6. Improving the model and validating its performance.

Learning objectives

After completing this module, students should be able to:

- Explain why an ML strategy is necessary.
- Understand human-level performance and its role in setting baselines.
- Explain when and how ML systems can surpass human-level performance.
- Describe transfer learning and multi-task learning with examples.
- Explain end-to-end deep learning, including its benefits and limitations.

2. Why ML Strategy?

2.1 Meaning

An ML strategy is a systematic approach for deciding what to improve in a machine learning system and in what order.

In traditional programming, developers explicitly specify rules. In machine learning, the system learns patterns from data. Therefore, improving an ML system requires careful decisions about data quality, model architecture, training procedures, and evaluation.

A project may have several possible improvements:

- Collecting more training data.
- Improving data labels.
- Increasing model size.

- Using a different neural network architecture.
- Applying regularisation.
- Changing the optimisation algorithm.
- Performing error analysis.

The main challenge is determining which action will improve the system most effectively.

2.2 Why is an ML strategy needed?

1. Clear objectives

Define the desired outcome and measurable performance target before development begins.

2. Appropriate evaluation metrics

Choose metrics such as accuracy, precision, recall, F1-score or mean squared error according to the problem.

3. Error analysis

Study incorrect predictions to identify whether the problem is caused by noisy data, insufficient training examples, model limitations or distribution differences.

4. Efficient use of resources

Reduce wasted experimentation, computational cost and development time.

2.3 The importance of train, development and test sets

A dataset is commonly divided into three parts.

Dataset	Purpose
Training set	Used to learn model parameters.
Development (validation) set	Used to compare models and tune hyperparameters.
Test set	Used for final, relatively unbiased evaluation after model selection.

The test set should not repeatedly guide model tuning, because this can cause the development process to overfit to the test data.

The data splits should represent the intended deployment environment. If training and test distributions differ significantly, a model may perform well during development but poorly in real-world use.

2.4 Bias and variance analysis

Two important causes of prediction error are high bias and high variance.

- High bias: The model is too simple or has not learned the underlying patterns sufficiently. It typically exhibits high training error.
- High variance: The model fits the training data well but generalises poorly to unseen examples. It typically exhibits a substantial gap between training and development performance.

Observation	**Possible problem**	**Potential action**
High training error and high development error	High bias or underfitting	Increase model capacity, improve features or train appropriately longer.
Low training error but high development error	High variance or overfitting	Add suitable data, use regularisation or simplify the model.
Good development performance but poor test performance	Test-set mismatch, leakage or repeated tuning	Audit the evaluation process and data distribution.

These are diagnostic clues, not universal rules; the correct action depends on the problem.

2.5 Error analysis

Error analysis is the systematic examination of incorrect model predictions to discover the most important sources of failure.

Example: Defective component detection

Suppose an image classifier makes 100 errors. After examining them, the developer finds:

- 40 errors due to poor lighting.
- 25 errors due to blurred images.
- 20 errors due to incorrect labels.
- 15 errors due to genuinely difficult cases.

This suggests that improving lighting variation in the training data and correcting labels may be more valuable than immediately replacing the neural network architecture.

Key principle: Measure the source of errors before deciding how to improve the model.

3. Understanding Human-Level Performance

3.1 Definition

Human-level performance is the level of accuracy or predictive performance achieved by a human performing the same task under a clearly defined evaluation protocol.

In deep learning, human-level performance is particularly useful in tasks where humans can recognise patterns effectively, such as image classification, speech recognition, and certain medical diagnosis tasks.

It provides a practical reference point for evaluating the progress of an ML model.

Examples include:

- Recognising handwritten digits.
- Identifying objects in photographs.
- Detecting defects in industrial equipment.
- Classifying spoken words.
- Detecting abnormalities in medical images.

Human performance is not necessarily perfect. Humans can make mistakes because of fatigue, limited information, ambiguous inputs and inconsistent judgement.

3.2 Bayes optimal error

Bayes optimal error is the minimum achievable prediction error under a given data distribution and loss function, even when the optimal decision rule is known.

It represents a theoretical lower bound on expected prediction error for that setting.

For example, suppose two medical images are visually indistinguishable but have different labels because of hidden factors unavailable in the images. No classifier using only those images can always predict the correct label.

This irreducible uncertainty contributes to the Bayes error.

Let:

- x = input features.
- y = actual class label.
- $P(y \mid x)$ = conditional probability of the label given the input.

For classification under zero-one loss, the Bayes-optimal classifier is

$$h^*(x) = \arg\max_y P(y \mid x)$$

The corresponding Bayes error is

$$E_{\text{Bayes}} = 1 - \mathbb{E}_x \left[\max_y P(y \mid x) \right]$$

This equation describes the minimum expected classification error when the true conditional distribution is known.

3.3 Comparing human, training and development errors

Consider a hypothetical image-classification task.

Illustrative classification error rates

Human-level error 2%

Training error 8%

Development error 10%

The training error is 6 percentage points higher than the human reference. The model may have a substantial avoidable learning gap, suggesting high bias or another training limitation.

The development error is 2 percentage points higher than the training error, indicating a generalisation gap that may reflect variance.

The developer should investigate both issues rather than assume that only one source of error exists.

3.4 Avoidable bias

Avoidable bias is the difference between the model's training error and an appropriate reference error, often human-level error when humans provide a reliable benchmark.

A simplified expression is

$$\text{Avoidable bias} \approx E_{\text{train}} - E_{\text{human}}$$

Using the preceding example,

$$\text{Avoidable bias} \approx 8\% - 2\% = 6\%$$

This is a diagnostic approximation, not a universal formal definition. Human error is only a suitable benchmark when the task, data and evaluation protocol are comparable.

If the model's error is close to human-level performance, reducing bias may become difficult, and the developer may need to focus more on variance, data distribution, or other limitations.

3.5 Why human-level performance is useful

Human-level performance helps developers:

1. Establish a meaningful baseline.
2. Estimate how much avoidable error remains.
3. Decide whether to focus on improving training or generalisation.

4. Identify tasks where human expertise can guide labelling and error analysis.
5. Prioritise experiments and resources.

Important limitation: Human-level performance is not automatically equal to Bayes optimal performance. Humans may perform worse than the theoretical optimum, and human performance may be difficult to measure reliably.

4. Surpassing Human-Level Performance

4.1 Meaning

Surpassing human-level performance means that a machine learning system performs better than a specified human benchmark on the same task and evaluation criteria.

Deep learning systems have surpassed human-level performance on certain narrowly defined benchmarks, especially where large datasets, consistent evaluation and repeated computation provide advantages.

However, this does not mean that the system has general intelligence or outperforms humans in every aspect of the task.

4.2 Why can an ML system outperform humans?

Several factors contribute to this capability.

1. Large-scale data processing: A model can learn from millions of labelled examples that no individual human could examine in full.

2. Consistency: A deployed model can apply the same decision rule repeatedly without ordinary human fatigue.

3. Pattern recognition: Deep neural networks can discover complex statistical relationships in high-dimensional data.

4. Specialised optimisation: Training algorithms adjust millions or billions of parameters to minimise a defined objective.

5. Narrow task specialisation: A model can be designed for one well-defined task and optimised specifically for its evaluation metric.

4.3 Example: Image classification

Suppose a quality-control system identifies defective electronic components.

System	Error rate

Human inspector benchmark	3%
ML model A	4%
ML model B	2%

Model A does not surpass the human benchmark because its error rate is higher. Model B does surpass the stated benchmark because its error rate is lower, assuming both were evaluated on the same representative dataset using the same criteria.

Lower error means better performance in this example.

4.4 Why is surpassing human-level performance difficult to measure?

Human and machine performance must be compared fairly.

- The same or appropriately matched test examples should be used.
- Human evaluators should receive suitable instructions and information.
- The evaluation metric must be identical.
- The test set should represent the intended real-world task.
- Statistical uncertainty should be considered, especially on small datasets.

For example, a medical AI system may outperform a human on a particular image-classification benchmark but still fail when images come from a different hospital or imaging device.

4.5 Human-level performance versus Bayes optimal performance

Human-level performance	**Bayes optimal performance**
Measured or estimated from human task performance	Theoretical optimum under a specified distribution and loss function
Can be affected by fatigue, experience and judgement	Assumes knowledge of the true conditional probabilities
Can be measured experimentally	Usually unknown in practical applications
Provides a practical benchmark	Provides a theoretical performance limit

4.6 Practical lesson

When human-level performance is a reliable benchmark, it helps establish a target. Once a model surpasses that benchmark, additional improvements may require more careful experimentation, better data, specialised expertise or improved evaluation methods.

But developers must not assume that beating humans on one benchmark means the model is universally superior or safe for deployment.

5. Transfer Learning

5.1 Definition

Transfer learning is a machine learning technique in which knowledge learned by a model on one task is reused to help solve another related task.

Instead of training a deep neural network entirely from random initialisation, a developer starts with a pretrained model and adapts it to a new problem.

Transfer learning is particularly useful when the target dataset is small, training resources are limited, or training a large model from scratch would be expensive.

5.2 Basic working principle

Source task

Large dataset: general images



Pretrained model

Learns edges, textures, shapes and higher-level features



Adaptation to target task

Replace or adapt the output layer; optionally fine-tune other layers



Target task

Classify defective and non-defective components

A model trained on a large collection of general images may already recognise useful visual features. These features can be reused to classify industrial components, reducing the amount of target-task training required.

5.3 Two common approaches

A. Feature extraction

In feature extraction, the pretrained network is used as a fixed feature extractor. Its learned weights are frozen, and a new classifier is trained on the target dataset.

Typical procedure:

1. Load a pretrained model.
2. Remove or replace its final classification layer.

3. Freeze the pretrained layers.
4. Train the new output layer on the target data.
5. Evaluate performance on validation data.

Advantages:

- Faster training.
- Fewer trainable parameters.
- Useful when the target dataset is small.

Limitation: The pretrained features may not be optimal for a substantially different target task.

B. Fine-tuning

In fine-tuning, some or all pretrained layers are allowed to update using the target dataset.

Typical procedure:

1. Load a pretrained model.
2. Replace its final layer for the new task.
3. Train the new layer.
4. Unfreeze selected pretrained layers.
5. Continue training with a suitably small learning rate.
6. Evaluate and select the best model using validation data.

Advantages:

- Can adapt the learned features to the target domain.
- Often improves performance when sufficient suitable target data are available.

Limitations:

- Requires more computation than fixed feature extraction.
- Can overfit small datasets.
- Excessively large learning rates may destroy useful pretrained representations.

5.4 Feature extraction versus fine-tuning

Feature	**Feature extraction**	**Fine-tuning**
Pretrained weights	Frozen	Some or all updated
Training cost	Usually lower	Usually higher
Target data requirement	Can work with very little data	Often benefits from more target data
Adaptability	Limited by fixed features	Greater adaptability

Risk of damaging pretrained features	Low because weights are frozen	Possible if training is poorly controlled

5.5 Example: Defect detection in electrical equipment

Suppose an engineering college has only 800 labelled photographs of defective and non-defective electrical components.

Training a large convolutional neural network from scratch may lead to overfitting.

A better starting approach could be:

- Select a pretrained image model.
- Replace its original output layer with a two-class classifier.
- Freeze the feature-extraction layers initially.
- Train the new classifier.
- Fine-tune selected deeper layers if validation results justify it.

The final performance must be evaluated on independent test data that represent the intended deployment conditions.

5.6 When should transfer learning be used?

Transfer learning is especially appropriate when:

- The target dataset is relatively small.
- The source and target tasks share useful features.
- Computational resources are limited.
- A suitable pretrained model is available.

It may be less effective when the source and target domains are substantially different or when the pretrained model's learned representations are unsuitable for the new task.

Key takeaway: Transfer learning reuses learned knowledge to reduce training effort and improve performance on a related target task.

6. Multi-task Learning

6.1 Definition

Multi-task learning (MTL) is a machine learning approach in which a single model learns to perform two or more related tasks simultaneously.

Instead of developing a completely independent model for each task, multi-task learning allows the tasks to share useful representations while maintaining task-specific outputs.

The underlying idea is that related tasks can help one another by learning common features from the same input data.

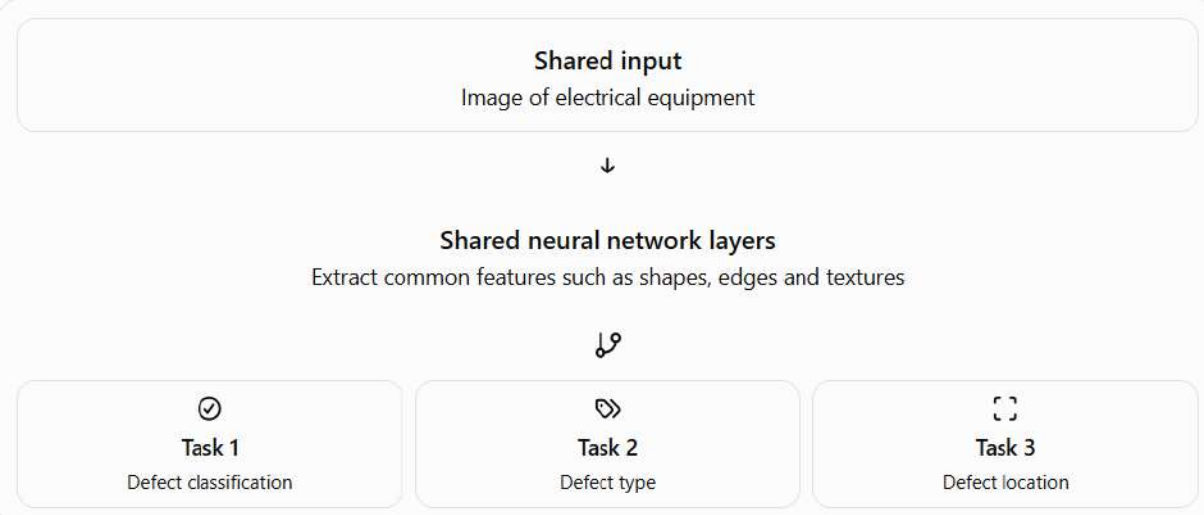
6.2 Example: Intelligent electrical equipment inspection

Consider a system that analyses photographs of electrical equipment. It is required to perform three tasks:

1. Classify whether the equipment is defective.
2. Identify the type of defect.
3. Estimate the location of the defect in the image.

These tasks are related because all three depend on visual features of the same equipment.

A multi-task neural network can learn shared image features and use separate output heads for the three tasks.



6.3 Architecture of multi-task learning

A typical multi-task neural network contains:

- Shared layers: Learn common representations from the input.
- Task-specific layers: Process shared representations for each task.
- Output heads: Produce the final prediction for each task.
- Joint loss function: Combines the individual task losses during training.

For example, a shared neural network may extract image features. One output head predicts whether a defect exists, another predicts the defect category, and a third estimates its position.

6.4 Mathematical formulation

Suppose a model performs T tasks. Let L_t represent the loss function for task t.

The joint objective can be expressed as

$$L_{\text{total}} = \sum_{t=1}^{T} \lambda_t L_t$$

where:

- T = number of tasks.
- L_t = loss associated with task t.
- λ_t = weight assigned to task t.
- L_{total} = combined training objective.

For three tasks,

$$L_{\text{total}} = \lambda_1 L_1 + \lambda_2 L_2 + \lambda_3 L_3$$

The weights can be selected to reflect task importance or to balance differences in loss scales. More sophisticated methods can learn or dynamically adjust these weights.

Example: If the three task losses are 0.20, 0.40 and 0.10, and all weights are 1, then

$$L_{\text{total}} = 0.20 + 0.40 + 0.10 = 0.70$$

The model is trained to reduce this combined loss.

6.5 Advantages of multi-task learning

1. Knowledge sharing: Related tasks can benefit from common learned features.
2. Improved generalisation: Shared representations may reduce overfitting when the tasks provide complementary information.
3. Computational efficiency: Shared layers can avoid duplicating the full model for each task.
4. Reduced model duplication: One network can produce several related predictions.
5. Better use of data: Labels available for different tasks may provide complementary supervision.

6.6 Limitations of multi-task learning

- Negative transfer can occur when tasks are poorly related.
- One task may dominate the combined loss.
- Task-specific data and labels may be incomplete.
- Training and evaluation become more complex.
- Improving one task can sometimes reduce performance on another.

6.7 Multi-task learning versus transfer learning

Feature	Transfer learning	Multi-task learning
Main objective	Reuse knowledge from a source task for a target task	Learn multiple tasks jointly
Training stages	Often sequential	Usually simultaneous or jointly optimised
Model structure	A pretrained model is adapted	Shared representation with multiple task outputs is common
Main benefit	Reduces the effort required to learn a new task	Shares information across related tasks
Example	Reuse a pretrained image model for defect detection	Predict defect type and location together

Important: The approaches are not mutually exclusive. A pretrained model can be fine-tuned using a multi-task learning architecture.

7. What Is End-to-End Deep Learning?

7.1 Definition

End-to-end deep learning is an approach in which a neural network learns a direct mapping from the original input to the desired output, replacing some or all manually designed intermediate processing stages.In a traditional pipeline, engineers may design separate stages for feature extraction, classification and post-processing. In an end-to-end system, a neural network can learn many of these transformations directly from training data.

7.2 Traditional machine learning versus end-to-end learning

Consider an automatic speech-recognition system.

Traditional pipeline



Audio signal

Hand-designed feature extraction

Acoustic model

Pronunciation and language processing

Recognised text

End-to-end pipeline

Audio signal

Neural network

↓

Learns the mapping from audio to text from training examples

↓

Recognised text

The end-to-end diagram is conceptual. Real systems may still contain tokenisation, decoding, signal processing, or other supporting components. The defining feature is that the main input-to-output mapping is learned jointly rather than relying on separately engineered intermediate predictors.

7.3 Mathematical representation

An end-to-end model can be represented as

$$\hat{y} = f_\theta(x)$$

where:

- x = original input.
- f_θ = neural network parameterised by θ.
- $\hat{y}$ = predicted output.

The parameters are learned by minimising a suitable loss function:

$$\theta^* = \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} L\left(f_\theta(x_i), y_i\right)$$

where:

- N = number of training examples.
- x_i = input of example i.
- y_i = corresponding target.
- L = loss function.
- θ^* = optimised model parameters.

The loss measures the discrepancy between predictions and target outputs. The specific loss function depends on the task.

7.4 Applications of end-to-end deep learning

1. Speech recognition

Input: Audio signal.

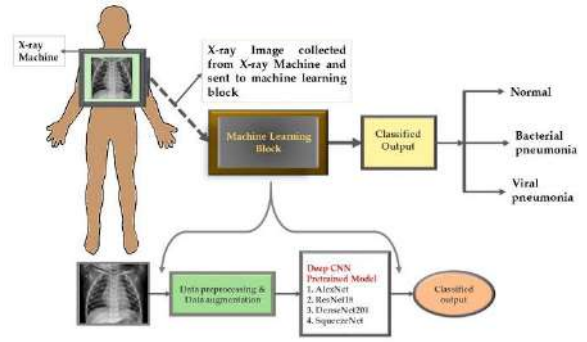
Output: Transcribed words.



2. Medical image analysis

Input: Medical image.

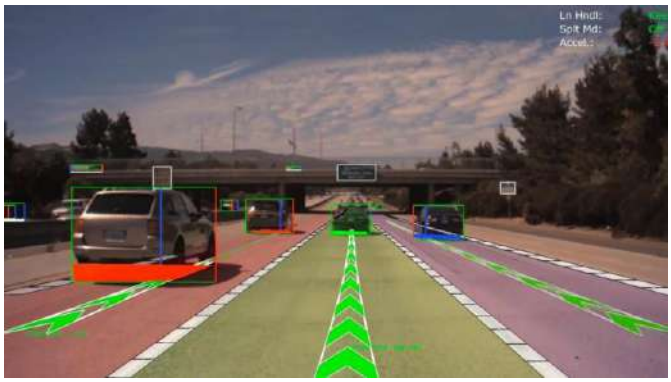
Output: Predicted class or abnormality score.



3. Autonomous driving perception

Input: Camera images or sensor data.

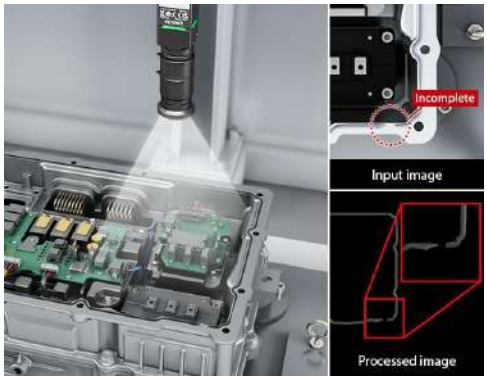
Output: Object detections, lane estimates or other driving-related predictions.



4. Industrial defect detection

Input: Component photographs.

Output: Defect class or defect localisation.



7.5 Advantages

- Reduces the need for manually engineered intermediate features.
- Allows intermediate representations to be optimised for the final objective.
- Can simplify a multi-stage learning pipeline.
- May achieve strong performance when sufficient representative training data are available.
- Enables the model to learn complex relationships between inputs and outputs.

7.6 Limitations

1. Large data requirements: Complex input-to-output mappings may require substantial labelled data.

2. Computational cost: Training large networks can require significant processing power and memory.

3. Reduced interpretability: Learned intermediate representations may be difficult to explain.

4. Limited modularity: Diagnosing and replacing individual stages can be more difficult.

5. Dependence on data quality: Incorrect labels, biased samples and distribution shifts can lead to poor predictions.

6. Difficulty with long or complex mappings: A carefully designed modular pipeline may be more practical when the task naturally requires explicit intermediate representations.

7.7 When should end-to-end learning be used?

It is suitable when:

- Sufficient representative input-output training data are available.
- The desired output can be defined clearly.
- A neural network can learn the mapping effectively.
- Computational resources are adequate.
- The benefits of joint optimisation outweigh the loss of modularity.

A hybrid approach may be better when domain knowledge, safety requirements or interpretable intermediate decisions are essential.

8. Comparison of the Six Major Concepts

Concept	Main idea	Example
Why ML Strategy	Decide how to improve an ML project systematically	Analyse errors before changing the model
Human-Level Performance	Establish a human benchmark	Compare defect-classification error with trained inspectors
Surpassing Human-Level Performance	Exceed a defined human benchmark	Model achieves 2% error versus human error of 3%
Transfer Learning	Reuse knowledge learned on another task	Fine-tune a pretrained image classifier
Multi-task Learning	Learn related tasks jointly	Predict defect type and location using shared features
End-to-End Deep Learning	Learn the main input-to-output mapping directly	Convert audio into text using a neural network

10. Important Questions with Answers

Q1. Why is an ML strategy important?

Answer: An ML strategy provides a systematic approach for improving a machine learning model. It helps developers select suitable evaluation metrics, analyse errors, identify bias and variance, choose appropriate experiments and use computational resources efficiently.

Q2. What is human-level performance? Explain avoidable bias.

Answer: Human-level performance is the performance achieved by humans on a specified task under a defined evaluation protocol. It can serve as a practical benchmark for an ML model.

Avoidable bias is approximately the difference between training error and an appropriate human-level reference error.

$$\text{Avoidable bias} \approx E_{\text{train}} - E_{\text{human}}$$

For example, if training error is 8% and human-level error is 2%, the approximate avoidable bias is 6 percentage points.

Q3. What is the difference between human-level performance and Bayes optimal performance?

Answer: Human-level performance is measured or estimated from human performance. Bayes optimal performance represents the theoretical minimum expected prediction error under a specified data distribution and loss function. Human performance may be worse than the theoretical optimum.

Q4. Explain transfer learning with an example.

Answer: Transfer learning reuses knowledge from a model trained on a source task to solve a related target task. For example, a convolutional neural network pretrained on general images can be adapted to detect defective electrical components. Its existing feature-extraction layers can be frozen initially, and the final classification layer can be trained using the target dataset.

Q5. What is multi-task learning? Write its loss function.

Answer: Multi-task learning trains a model to perform multiple related tasks simultaneously using shared representations and task-specific outputs.

The combined loss is

$$L_{\text{total}} = \sum_{t=1}^{T} \lambda_t L_t$$

where L_t is the loss of task t and λ_t is its loss weight.

Q6. Differentiate between transfer learning and multi-task learning.

Answer: Transfer learning uses knowledge learned from a source task to improve a target task, often through sequential training. Multi-task learning jointly trains a model on several related tasks, allowing them to share useful representations. Both techniques can be used together.

Q7. What is end-to-end deep learning? State its advantages and limitations.

Answer: End-to-end deep learning trains a neural network to map the original input directly to the desired output, learning the necessary intermediate representations.

Advantages include reduced manual feature engineering, joint optimisation and potentially strong predictive performance.

Limitations include high data and computational requirements, reduced interpretability and sensitivity to data quality.

Q8. Explain end-to-end deep learning using an engineering application.

Answer: In electrical equipment inspection, an end-to-end model receives a photograph of a component and predicts whether it is defective. The neural network learns useful image representations and the final classification mapping from labelled examples. The system is evaluated on independent test data to determine its performance on unseen components.

Short Questions with Answers

A. Why ML Strategy?

Q1. What is an ML strategy?

Answer: An ML strategy is a systematic approach for developing, evaluating and improving a machine learning model.

Q2. Why is an ML strategy important?

Answer: It helps identify model weaknesses, select suitable improvements and reduce development time and computational cost.

Q3. What is a training set?

Answer: A training set is the dataset used to learn the parameters of a machine learning model.

Q4. What is a development set?

Answer: A development set, also called a validation set, is used to compare models and tune hyperparameters.

Q5. What is a test set?

Answer: A test set is used to evaluate the final model's performance on unseen data.

Q6. What is error analysis?

Answer: Error analysis is the process of examining incorrect predictions to identify the main causes of model failure.

Q7. What is high bias?

Answer: High bias occurs when a model is too simple or insufficiently trained to learn the underlying patterns, leading to underfitting.

Q8. What is high variance?

Answer: High variance occurs when a model fits training data well but performs poorly on unseen data, often due to overfitting.

B. Human-Level Performance

Q9. What is human-level performance?

Answer: It is the level of performance achieved by humans on a specific task under a defined evaluation procedure.

Q10. What is Bayes optimal error?

Answer: Bayes optimal error is the minimum achievable expected prediction error under a specified data distribution and loss function.

Q11. What is avoidable bias?

Answer: Avoidable bias is approximately the difference between training error and an appropriate reference error, such as human-level error.

Q12. Why is human-level performance useful in ML?

Answer: It provides a practical benchmark for evaluating a model and identifying opportunities for improvement.

C. Surpassing Human-Level Performance

Q13. What is meant by surpassing human-level performance?

Answer: It means that an ML model performs better than a specified human benchmark on the same task and evaluation criteria.

Q14. How can an ML model outperform humans?

Answer: It can learn from large datasets, identify complex patterns and perform specialised tasks consistently.

Q15. Does surpassing human-level performance mean that AI is always better than humans?

Answer: No. It means the model performs better on a particular task or benchmark, not necessarily in every situation.

D. Transfer Learning

Q16. What is transfer learning?

Answer: Transfer learning reuses knowledge learned by a model on one task to improve its performance on a related task.

Q17. Why is transfer learning useful?

Answer: It can reduce training time, computational requirements and the amount of labelled target data needed.

Q18. What is a pretrained model?

Answer: A pretrained model is a model that has already been trained on a dataset before being adapted to a new task.

Q19. What is feature extraction in transfer learning?

Answer: Feature extraction uses the pretrained model to generate features while keeping its learned weights fixed, then trains a new classifier.

Q20. What is fine-tuning?

Answer: Fine-tuning means updating some or all pretrained model weights using data from the target task.

E. Multi-Task Learning

Q21. What is multi-task learning?

Answer: Multi-task learning trains a model to perform two or more related tasks simultaneously.

Q22. What are shared layers?

Answer: Shared layers are neural network layers that learn common features used by multiple tasks.

Q23. What is a task-specific output layer?

Answer: It is a layer designed to produce predictions for a particular task.

Q24. Write the loss function for multi-task learning.

Answer:

$$L_{\text{total}} = \sum_{t-1}^{T} \lambda_t L_t$$

Here, L_t is the loss for task t, and λ_t is its corresponding weight.

Q25. State two advantages of multi-task learning.

Answer: It enables knowledge sharing between related tasks and may improve generalisation while reducing duplicated model components.

Q26. What is negative transfer?

Answer: Negative transfer occurs when knowledge sharing between tasks reduces performance instead of improving it.

F. End-to-End Deep Learning

Q27. What is end-to-end deep learning?

Answer: It is an approach in which a neural network learns the main mapping from original inputs to desired outputs.

Q28. Give two applications of end-to-end deep learning.

Answer: Speech recognition and image classification are two common applications.

Q29. State two advantages of end-to-end deep learning.

Answer: It reduces manual feature engineering and enables joint optimisation of the input-to-output mapping.

Q30. State two limitations of end-to-end deep learning.

Answer: It may require large amounts of labelled data and significant computational resources.

Q31. Write the general mathematical representation of an end-to-end model.

Answer:

$$\hat{y} = f_\theta(x)$$

Here, x is the input, f_θ is the neural network, and $\hat{y}$ is the predicted output.

G. Important Comparison Questions

Q32. Differentiate between transfer learning and multi-task learning.

Answer: Transfer learning reuses knowledge from a source task for a target task, whereas multi-task learning trains on multiple related tasks jointly.

Q33. Differentiate between human-level and Bayes optimal performance.

Answer: Human-level performance is measured from human performance, while Bayes optimal performance represents the theoretical minimum expected prediction error.

Q34. What is the difference between feature extraction and fine-tuning?

Answer: Feature extraction keeps pretrained weights fixed, whereas fine-tuning updates some or all pretrained weights.

Q35. Give one engineering application of multi-task learning.

Answer: An electrical equipment inspection system can simultaneously identify whether a component is defective, classify the defect and estimate its location.

Q36. Give one engineering application of transfer learning.

Answer: A pretrained image classification model can be adapted to detect defects in electrical components using a relatively small labelled dataset.